WWW.SCIAM-MAGAZINE.NAROD.RU



# Воспроизведение «живых» сцен на экране дисплея: задачи для начинающих программистов

А. К. ДЬЮДНИ

ЭФФЕКТИВНЫЕ кинофокусы не всегда требуют сложного технического оборудования или финансовой поддержки крупных киностудий. Любители фантастических сюжетов теперь сами могут воспроизводить удивительные явления, хотя и ограниченного репертуара, на экранах своих домашних компьютеров с помощью того, что можно было бы назвать программами видеоэффектов. Представим на минуту, что по экрану компьютера внезапно поползли червяки. Программа видеоэффектов способна создать такую «живую» сцену. Или, например, экран может превратиться в водяную поверхность с расходящимися кругами от падающих дождевых капель. Создать этот эффект тоже в общем-то довольно просто. И наконец, вообразим, что наш экран — это иллюминатор, через который видна Вселенная с взрывающимися или стремительно пробегающими мимо звездами, как будто мы находимся на борту космического корабля, с колоссальной скоростью пролетающего по звездному коридору. Подобная иллюзия достигается при помощи третьей, довольно экономной программы.

Экономия средств, на мой взгляд, одно из главных свойств программ подобного рода. Если достигается какой-то поразительный эффект, то лучше, если порождающая его программа оказывается чрезвычайно короткой. Как же лучше всего ввести читателей в таинственный мир удивительных явлений, видимых на экране дисплея и скрывающихся за ним. Судя по сравнительно небольшой, но всегда присутствующей части получаемых мною писем, многие начинающие любители компьютерных игр с грустью смотрят на страницы статей нашей рубрики, чувствуя свою неспособность превратить представленные в них алгоритмы в работающие программы. Они остаются лишь пассивными зрителями, когда мимо них в ярком парадном шествии проходят множества Мандельброта, лестницы слов, сотворенные компьютерами карикатуры, сосуды с первобытным бульоном и звездные скопления. Они могут лишь мечтать о той радости, которую испытывают авторы программ, видя, как алгоритмы обретают плоть и кровь и оживают на экране дисплея.

Поэтому на этот раз я решил помочь таким читателям и не ограничиваться лишь общим описанием приемов, с помощью которых можно воспроизводить упомянутые выше видеоэффекты. Я расскажу обо всех «винтиках и гаечках», которые превращают алгоритм в программу. Я объясню, как населить экран ползающими червяками, не только на универсальном алгоритмическом языке, но и на конкретном языке программирования — Бейсик. Не ограничиваясь объяснением того, как нужно строить сам алгоритм, я приведу пример преобразования, который поможет читателям лучше понять как прошлые, так и будущие статьи нашей рубрики. Возможно, начинающие программисты почувствуют, что в них самих тоже произошло важное преобразование.

Кто бы смог удержаться от соблазна написать программу, порождающую на экране симпатичных копошащихся червячков? Энтузиазм читателей, наверное, не очень пострадает, когда они узнают, что каждый червяк на экране представляет собой не что иное, как цепочку окружностей. Впрочем, чтобы составить программу для какого-нибудь видеоэффекта (и вообще любую программу), нужно на время абстрагироваться от эмоций и сосредоточиться на анализе. Каким же образом заставить червяка двигаться? Добавляя новую окружность у головы и удаляя старую у хвоста: в результате появляется ощущение движения, как будто бы червяк ползет, — конечно, это иллюзия, но фокус тем не менее удается.

Идея программы WORMS (червяки) зародилась у меня, когда я впервые увидел подобный эффект на экране моего новенького компьютера SUN. Кто-то занес эффектную графическую программу в каталог системных и прикладных программ, обычно поставляемых в комплекте с машиной. Этот кто-то, по-видимому, подумал, что зрелище копошащихся на большом экране дисплея SUN червей убедит покупателей, что высокая цена на машину вполне оправдана.

Действительно, решение написать эту программу пришло, когда я задумался над тем, каким образом достигается впечатление, что червяк ползет. Пристально вглядываясь в экран, я вскоре понял, что эффект движения достигается каждый раз за счет добавления новой окружности к голове червяка и стирания старой в хвосте. «Ага, — сказал я себе, — это просто». Я сразу представил себе коротенькую программку, но затем подумал о том, что программа, стирающая старую окружность, должна помнить, где находится эта окружность. А поскольку каждое звено червяка рано или поздно нужно будет стереть, положение каждой окружности нужно запоминать.

Простейшая структура данных, позволяющая запоминать большое число однородных элементов данных, это массив. Следовательно, теперь нужно было подумать о том, как можно запоминать окружности в массиве. На каком бы языке мы ни писали программу, графическая команда «нарисовать окружность» (или стереть ее, проведя по ее контуру черной линией) обычно требует указания двух координатных значений, а именно центра окружности и радиуса. Радиус всех окружностей, из которых строятся звенья червяков, будет постоянной величиной, а вот координаты центра должны меняться от одной окружности к другой. Поэтому необходимо хранить лишь координаты центров. Так как, вообще говоря, абсцисса центра отличается от ординаты, то координатные значения центров удобнее хранить не в одном, а в двух массивах. Назовем их *xcirc* и *ycirc*. Таким образом, элемент *xcirc*(*i*) будет хранить координату *i*-й окружности по оси *x*, а *ycirc*(*i*) — по оси *y*.

Теперь мы подошли к главной трудности в программе WORMS: нужно, чтобы она каким-то образом добавляла к массиву новые окружности и удаляла из него старые. Подобно червяку, сами массивы будут состоять из цепочек окружностей. Однако массивы не движутся по экрану. Мы заведем специальную переменную, называемую указателем, в которой будет храниться та позиция массива, где в данный момент содержится координата последней, хвостовой

WWW.SCIAM-MAGAZINE.NAROD.RU



окружности червяка. И в математике, и в программировании чудодейственной силой обладает простая процедура — присвоение имени. Назвав переменную-указатель *tail* (хвост), я сразу же догадался, какой должна быть главная, ключевая команда в программе WORMS:

$$tail \leftarrow tail + 1.$$

Другими словами, на каждом шаге основного цикла программы указатель хвоста будет продвигаться в массиве на одну позицию.

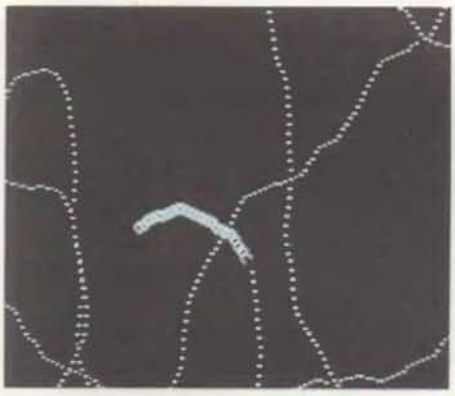
Если, например, последнее значение *tail* было равно 7, то координаты хвостовой окружности червяка хранились соответственно в элементах *xcirc* (7) и *ycirc* (7). Когда мы выполним команду приращения, приведенную выше, новое значение переменной *tail* будет равно 8. Тем временем мы должны будем нарисовать новую окружность и поместить ее координаты в массивы *xcirc* и *ycirc*. Логично поместить их на ту позицию, которую в данный момент занимают координаты старого хвоста, а именно позицию 7.

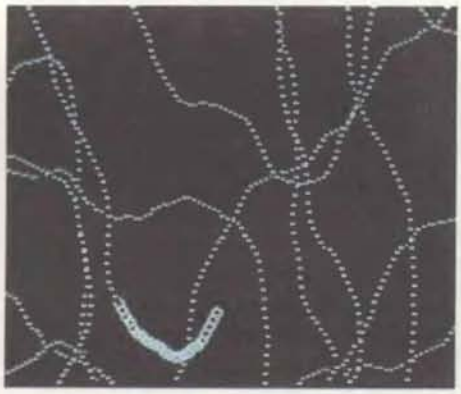
Пока что я не имею ни малейшего понятия, каким образом программа должна определять то место, где нужно рисовать новые головные окружности на экране, — это отдельный вопрос, к которому мы перейдем позже. Часто трудности при составлении программы объясняются той путаницей, которая возникает в голове, когда мы хотим учесть сразу много различных действий.

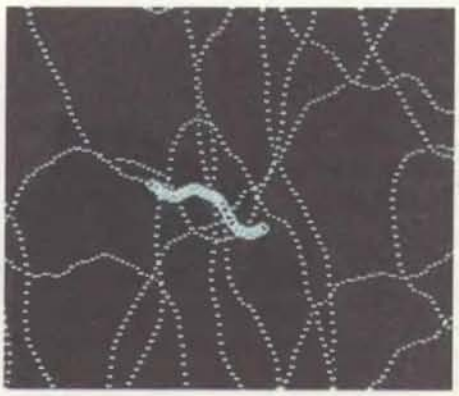
Еще не приступая к построению наброска алгоритма программы WORMS на бумаге, я уже принял решение, что нужно завести два массива, хранящие координаты всех окружностей, составляющих на данный момент тело червяка. Указатель *tail* должен продвигаться вдоль каждого массива, заменяя координаты старой хвостовой окружности координатами новой головной. Однако что произойдет, когда указатель *tail* достигнет конца массива? Конечно же, он должен снова перескочить на начало, для чего нужно воспользоваться вычислениями по некоторому модулю. Если, скажем, червяк состоит из 10 колец и старое значение указателя *tail* равно 9, то его новое значение должно быть равно 0. Пока наш алгоритм имеет следующий вид:

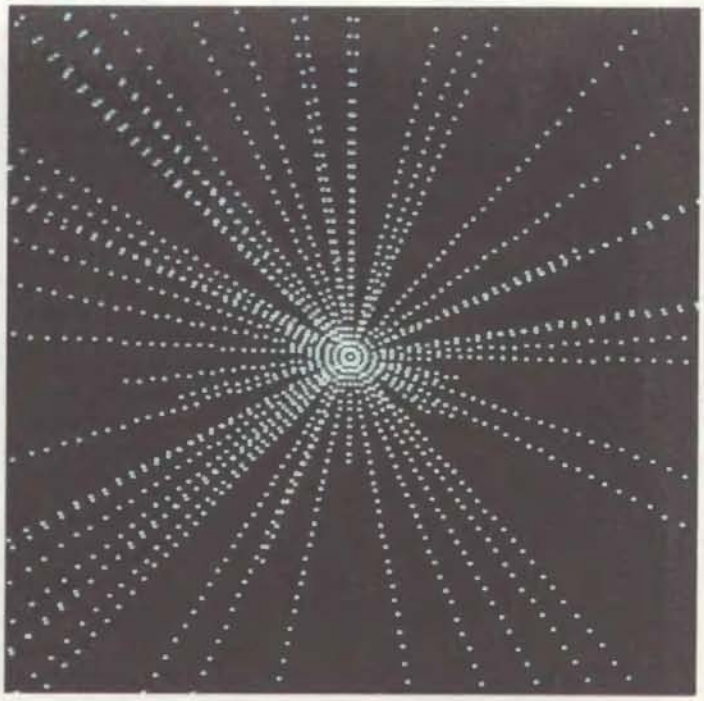
```
цикл
  tail ← tail + 1 (по какому-то модулю)
    выбрать координаты новой
    головы
    подставить в xcirc (tail) и ycirc
    (tail)
    нарисовать новую голову
    стереть хвост
конец цикла
```

Главная польза алгоритмического подхода к построению компьютерной программы заключается как раз в неопределенности и гибкости алгоритмического языка. В любом месте алгоритма можно написать что угодно, лишь бы это имело какой-то потенциальный смысл, который можно уточ-









*Червяк, оставляющий след* (вверху), *дождевые капли* (слева внизу) *и звездный взрыв* (справа внизу)

WWW.SCIAM-MAGAZINE.NARODU.RU



нить позже. Например, сейчас я пока отложил вопрос о том, как программа должна определить координаты новой головы, просто записав «выбрать координаты новой головы». В подробном алгоритме это, возможно, превратится в несколько строк. Но даже при такой грубой детализации, подставив пока эту фразу, я сразу же яснее представил себе, какой должна быть структура программы. Я даже нашел ошибку: как программа сможет «стереть хвост», если она не знает, где он находится? Дело в том, что выше в теле цикла на место этих координат были записаны координаты новой головы. Стереть старый хвост следовало раньше, до того как на место старых были подставлены новые координаты. Я внес в алгоритм соответствующие коррективы. Возможно, читатели захотят сами внести исправления, прежде чем посмотреть, как это было сделано у меня.

Теперь настало время сделать второй проход по алгоритму. Какой именно цикл я подразумевал, заключив в него весь остальной алгоритм? Пытаясь ответить на этот вопрос, я сразу же натолкнулся на другую проблему: как должна завершаться программа? Сначала я подумал о простейшей схеме с фиксированным числом повторений, или итераций, реализуемой посредством оператора цикла FOR, в котором некоторый индекс автоматически отсчитывает несколько тысяч итераций, в то время как червяк будет ползать по экрану. Но лучше было бы закончить карьеру червяка, когда будет нажата какая-нибудь клавиша. Таким образом, программа будет снова и снова выполнять цикл, пока человек не нажмет определенную клавишу.

Следует также уточнить, что конкретно подразумевается под словами «по какому-то модулю» в первом операторе цикла. Этот модуль должен быть равен размеру массива. Однако тут возникает еще один вопрос, какой длины будет червяк? Я произвольно выбрал 25 звеньев, решив, что червяк такой длины будет выглядеть достаточно правдоподобно.

Теперь алгоритм программы WORMS принял уже следующий вид (здесь mod означает «по модулю»):

```
repeat
    tail ← tail + 1 (mod 25)
    стереть хвост
    выбрать координаты новой
    головы
    подставить в xcirc (tail) и ycirc
    (tail)
    нарисовать новую голову
    until нажатие клавиши
```

Продолжая уточнять алгоритм, я решил заменить фразу «стереть хвост» чем-то более близким к тому языку программирования, которым я намереваюсь в конце концов воспользоваться. Поскольку я хотел стереть старую окружность, нарисовав черный круг там, где раньше был белый (на моем нормально черном экране), я подставил выражение «черная окружность в *circ* (*tail*)», что означает черную окружность с центром в точке с координатами *xcirc* (*tail*) и *ycirc* (*tail*).

Теперь я приступаю к фразе «выбрать координаты новой головы». Посредством какого механизма программа может решить, где следует нарисовать новую головную окружность? Очевидно, она должна быть в непосредственной близости от предыдущей и где-то «впереди» от нее. Более того, чтобы извивания и повороты червяка выглядели естественно, новое направление не должно слишком резко отличаться от предшествующего. Здесь решение проблемы заключалось просто в том, чтобы ввести одну-единственную новую переменную *dir* (от слова *direction* — направление). Она должна содержать текущее направление движения червяка, выраженное в градусах. При каждом повторении главного цикла программа будет произвольно либо увеличивать, либо уменьшать значение *dir* на 10°.

Пришло время решить, насколько близко друг от друга должны быть окружности и какого они должны быть размера. В качестве пробного значения радиуса можно принять, например, 4. Чтобы червяк выглядел как единое целое, нужно, чтобы окружности частично перекрывали друг друга. Далее я заменил фразу «выбрать координаты новой головы» следующим фрагментом:

```
change ← случайное число
if change < 0,5
   then dir ← dir + 10
   else dir ← dir − 10
x ← xcirc (tail − 1)
y ← ycirc (tail − 1)
newx ← x + 4 · cos (dir)
newy ← y + 4 · sin (dir)
```

В этом алгоритмическом фрагменте случайным образом выбирается число, присваиваемое переменной *change* (изменение). Большинство языков программирования имеет оператор выбора случайного числа в диапазоне от 0 до 1. Поскольку случайное число этого диапазона будет меньше 0,5 в среднем в 50% случаев, переменная *dir* будет увеличиваться в среднем через раз. Конечно, эта статистика справедлива лишь при большом количестве испытаний. На коротких же промежутках времени переменная *dir* будет меняться непредсказуемым образом в ту или иную сторону на 10°. Переменные *x* и *y* — это просто координаты предыдущей головной окружности. Последние два оператора представляют собой элементарные тригонометрические вычисления, при помощи которых мы получаем изменения *x* и *y*, приведенные к радиусу, равному 4. Таким образом, координаты новой головы выбраны.

Осталось лишь сделать еще два шага, чтобы окончательно уточнить алгоритм. Запомнить координаты новой головы в элементах *xcirc* (*tail*) и *ycirc* (*tail*) можно очень просто, при помощи двух операторов присваивания. Чтобы нарисовать новую голову, можно воспользоваться графической псевдокомандой, при помощи которой мы ранее стирали хвост.

Наконец, нужно инициализировать (т. е. присвоить начальные значения) массивы и переменные перед выполнением алгоритма. В данном случае указателю *tail* присвоим значение 1, а элементам массивов *xcirc* (1) и *ycirc* (1) — по 100 каждому. Остальным элементам этих массивов автоматически присвоены нулевые значения. Переменная направления *dir* имела исходное значение 0. Таким образом, червяк «начинался» в точке с координатами (100, 100) и двигался под углом 0 к горизонту. Точка (100, 100) лишь условно обозначает координаты центра окружности; на экране в зависимости от типа компьютера эти координаты могут быть иными.

Одна проблема все же осталась: а что, если червяк уползет за пределы экрана? Чтобы решить ее, я наделил программу еще одним свойством. Если координатные значения *x* или *y* выходят за диапазон, соответствующий размерам экрана, то они автоматически преобразуются в координаты точки, расположенной на его противоположном крае. Предположив, что наш гипотетический экран имеет диапазон координатных значений от 0 до 199, мы можем воспользоваться следующими операторами, преобразующими координаты так, чтобы они всегда находились в пределах выбранного диапазона:

```
newx ← newx mod 200
newy ← newy mod 200
```

Возможность вычисления по заданному модулю имеется практически во всех языках программирования. При этом преобразуемое число (*newx*) делится на заданный модуль (200) и в качестве результата берется остаток от деления. Теперь переменная, которой присваивается остаток (*newx*) имеет значение, вычисленное по модулю.

Ну вот и все, что касается програм-

WWW.SCIAM-MAGAZINE.NAROD.RU



мы WORMS. Или может быть правильнее было бы назвать ее WORM (червяк)? Ведь алгоритм порождает лишь одного червяка. Чтобы к названию программы по праву добавить окончание S, читатели могут завести по одному набору массивов и переменных для каждого червяка, который должен появиться на экране. Рассмотренный выше алгоритм полностью приведен на рисунке внизу. Рядом приведена эквивалентная программа на языке Бейсик. Как педагог я был бы весьма разочарован, узнав, что кто-нибудь из наших читателей просто-напросто скопировал мою программу на Бейсике в свою машину, не усвоив объяснений того, о чем так подробно говорилось выше.

Сравнив строка за строкой алгоритм и программу, мы обнаружим довольно близкое сходство между ними. Действительно, два основных блока операторов в алгоритме практически не претерпели никаких изменений при переводе на Бейсик (для машины IBM PC Бейсик, версия 3.0). Основные различия связаны со свойствами реализации ввода с клавиатуры в языке Бейсик IBM PC, а также отсутствием цикла типа *repeat* в данной версии Бейсика.

В первом случае KEY (1) (функциональная клавиша F1 на клавиатуре компьютера IBM PC) была выбрана в качестве выключателя, на который пользователь программы может нажать, когда ему надоест наблюдать за тем, как червяк ползает по экрану. В строке 70 программы эта клавиша активизируется, сообщая системе, реализующей команды Бейсика, что нужно постоянно проверять, не поступил ли сигнал от клавиши F1. В строке 90 команда KEYOFF удаляет пояснительный текст в нижней части экрана, чтобы освободить место для червяков. Однако главная команда, использующая клавишу F1, содержится в строке 60. Здесь программе надлежит совершить переход на строку 290, если пользователь нажал на клавишу F1. В строке 290 экран очищается и выполнение программы завершается командой END.

Во втором случае ввиду отсутствия в языке цикла типа *repeat* было необходимо построить эквивалентную последовательность выполнения операторов с помощью команды перехода GOTO на строку 280 программы. В результате программа будет снова и снова возвращаться на строку 110, пока кто-нибудь не нажмет на клавишу F1. Тело цикла выделено в программе двумя строками комментария под номерами 100 и 270. Эти комментарии были вставлены для того, чтобы читатели лучше видели структуру цикла в программе.

Есть и другие отличия между алгоритмом и программой, связанные со свойствами языка Бейсик. В строке 10 устанавливается размер массивов, равный 25 элементам для каждого массива. В строке под номером 80 устанавливается высокая разрешающая способность экрана для компьютера марки IBM PC. Здесь точки изображения располагаются очень близко друг к другу. В строке 110 новая переменная *wastail* принимает значение переменной *tail* непосредственно перед тем, как последняя получает приращение. Следовательно, переменную *wastail* можно подставить в алгоритме вместо выражения *tail* — 1 там, где вычисляются значения $x$ и $y$. Однако главная причина введения переменной *wastail* заключается в том, что *tail* как индекс массива не может принимать нулевое значение в данной версии языка Бейсик. В связи с этим в строке 120 единица прибавляется после того, как взят модуль, но не прежде. Записывая алгоритм, я допустил неточность, небрежно предположив, что индекс *tail* должен пробегать значения от 0 до 24. Реальная команда,

```
xcirc(1) ← 100
ycirc(1) ← 100
dir ← 0
tail ← 1
repeat
    tail ← (tail + 1) mod 25
    черный кружок в circ(tail)
    change ← случайное число
    if change <
      then dir ← dir + 10
      else dir ← dir − 10
    x ← xcirc(tail − 1)
    y ← ycirc(tail − 1)
    newx ← x + 4 · cos (dir)
    newy ← y + 4 · sin (dir)
    newx ← (newx + 100) mod 200 − 100
    newy ← (newy + 100) mod 200 − 100
    xcirc(tail) ← newx
    ycirc(tail) ← newy
    белый кружок в (newx, newy)
until любая клавиша
```

```
10  DIM XCIRC(25), YCIRC(25)
20  XCIRX(1) = 100
30  YCIRC(1) = 100
40  DIR = 0
50  TAIL = 1
60  ON KEY(1) GOSUB 290
70  KEY(1) ON
80  SCREEN 2
90  KEY OFF
100 повторить
110   WASTAIL = TAIL
120   TAIL = (TAIL MOD 25) + 1
130   CIRCLE (XCIRC(TAIL), YCIRC(TAIL)),4,0
140   CHANGE = RND
150   IF CHANGE < .5 THEN DIR = DIR + .1745 ELSE DIR = DIR − .1745
160   X = XCIRC(WASTAIL)
170   Y = YCIRC(WASTAIL)
180   NEWX = X + 4 * COS(DIR)
190   NEWY = Y + 4 * SIN(DIR)
200   NEWX = NEWX MOD 600
210   NEWY = NEWY MOD 200
220   IF NEWX < 0 THEN NEWX = NEWX + 600
230   IF NEWY < 0 THEN NEWY = NEWY + 200
240   XCIRC(TAIL) = NEWX
250   YCIRC(TAIL) = NEWY
260   CIRCLE(NEWX, NEWY),4,1
270 пока не будет нажата клавиша
280 GOTO 110
290 SCREEN 0
300 END
```

*Превращение алгоритма* WORMS (червяки) *в программу на языке Бейсик*

WWW.SCIAM-MAGAZINE.NAROD.RU



при помощи которой рисуется окружность на языке Бейсик IBM PC, содержится в строке 130. Центр окружности находится в точке с координатами XCIRC (TAIL), YCIRC (TAIL), радиус равен 4, а цвет обозначен цифрой 0 (черный).

В строке 150 значение переменной DIR получает приращение не в 10°, а на эквивалентную величину 0,1745 рад. (В версии Бейсика для компьютера IBM PC углы не измеряются в градусах.) В строках 200 и 210 переменные NEWX и NEWY ограничиваются значениями 600 и 200 соответственно. Это делается для того, чтобы координаты не выходили за пределы экрана для компьютера IBM PC. Средство Бейсика, позволяющее выделять модуль, хорошо работает для положительных чисел, но не работает с отрицательными. Поэтому, когда червяк пытается уползти с экрана в отрицательном направлении, модуль берется с прибавлением соответствующего размера экрана, так как это сделано в строках 220 и 230. Червяк, уползающий с одного края прямоугольника, появляется сразу же на другом краю. В строке 290 команда SCREEN 0 очищает экран для возможных в дальнейшем текстов.

Читатели, которые пожелают усовершенствовать свою версию программы, могут вставить перед началом цикла команду RANDOMIZE, не позволяя, таким образом, червяку повторять тот же путь. Можно также предусмотреть возможность повторного запуска программы после ее остановки. Можно, например, трассировать путь червяка пунктирной линией из точечек, подобно тому, как мы сделали это на верхнем рисунке, приведенном на с. 83.

Программы для других уже упомянутых видеоэффектов теперь, наверное, могут быть написаны даже новичками, если они к тому же воспользуются некоторыми подсказками. Программа, которую я назвал RAINDROPS (дождевые капли), случайным образом выбирает точку на экране дисплея и описывает вокруг нее последовательность концентрических окружностей возрастающего радиуса. Затем программа выбирает другую точку и повторяет процедуру построения окружностей. RAINDROPS продолжает работать до тех пор, пока «хозяин погоды» не нажмет клавишу. Таким образом, программа строится из двух циклов: внешнего цикла, в котором выбирается точка, и внутреннего цикла, в котором рисуются окружности вокруг выбранной точки. Программа преобразует два случайных числа в диапазоне между 0 и 1 в координаты точки на экране путем умножения этих чисел на соответствующие размеры экрана. Получаемая в результате точка ($x$, $y$) становится центром последовательности окружностей, рисуемых во внутреннем цикле. Пусть индекс $k$ пробегает значения от 1 до 25 по мере того, как на экране строятся белые (или какого-нибудь другого цвета) окружности. Центры всех этих окружностей находятся в точке с координатами ($x$, $y$), а радиусы принимают значения, равные $4k$. Проще, конечно, оставлять нарисованные таким образом окружности неподвижными. Однако эффект был бы более реалистичным, если бы несколько окружностей, или даже одна, распространялись, подобно волне. Для этого программа RAINDROPS должна стирать (перерисовывать черным цветом) окружность радиуса $4(k - 1)$ всякий раз после того, как нарисована окружность радиуса $4k$.

Программа STARBURST (звездный взрыв) демонстрирует эффект космического взрыва. По существу тот же эффект можно интерпретировать как полет вдоль огромного звездного коридора, наблюдаемого из иллюминатора космического корабля, который движется с околосветовой скоростью. Звезды каждый раз перемещаются на небольшое расстояние вдоль линий перспективы по направлению к наблюдателю. Величина сдвига возрастает по мере того, как звезды приближаются к наблюдателю. Создается впечатление, что объекты, находящиеся ближе, движутся с большей скоростью по сравнению с удаленными объектами. Программа STARBURST несколько сложнее, чем RAINDROPS. Она должна постоянно следить за всеми звездами, находящимися в пределах экрана. Для этого читатели могут воспользоваться массивом, назвав его, скажем, *stars* (звезды) — а правильнее будет сказать двумя массивами, как в программе WORMS, — чтобы хранить координаты $x$ и $y$ каждой звезды. Координата $z$ оказывается скрытой в кажущемся движении звезд.

Когда звезда впервые появляется в исходной точке (100, 100), ее перемещения вначале очень малы, но затем они растут по мере того, как звезда приближается к наблюдателю (или наоборот — наблюдатель приближается к звезде). Величина приращения обратно пропорциональна $z$ — кажущемуся расстоянию от наблюдателя. При составлении программы можно запланировать, например, по 25 сдвигов на каждую звезду, прежде чем она выйдет за пределы экрана. Таким образом, расстояние $z$ до любой данной звезды можно принять равным $25 - k$, где $k$ — количество итераций в расчете кажущихся расстояний до этой звезды.

В большом цикле программы на каждом шаге создается одна новая звезда, а для всех остальных звезд координаты, хранящиеся в массивах, получают приращения.

Я НЕ ПЕРЕСТАЮ удивляться изобретательности наших читателей. В статье, опубликованной в октябрьском номере журнала, я посоветовал читателям, не располагающим компьютерами, не утруждать себя решением некоторых проблем, возникающих в игре со словами, и с удивлением обнаружил, что именно эти читатели нашли интересные решения.

В частности, был сформулирован вопрос о возможности преобразования одного $n$-буквенного слова в другое при помощи цепочки промежуточных слов. При переходе от одного промежуточного слова к другому разрешалось заменять лишь одну букву. В результате получается лестница слов. Например, в случае $n = 2$ все двухбуквенные слова английского языка (по словарю, которым я пользовался) оказались связанными друг с другом лестницами слов. Я оставил открытым вопрос о том, связаны ли все $n$-буквенные слова между собой для бóльших значений $n$. Оказалось, что для исследования связей между словами можно вполне обойтись и без компьютера.

В качестве решения трехбуквенной задачи многие читатели предложили слово *gnu*. А для $n = 5$ несколько человек назвали слово *xylem*. Решение задачи заключалось в том, чтобы найти слово, которое невозможно соединить ни с одним другим словом. Если такое слово нашлось, то общий вопрос задачи имеет отрицательный ответ. Согласно исследованиям Дж. Миллера из Сан-Франциско, в английском языке существует 1217 слов, с которыми невозможно связать слово *horse* (лошадь), приведенное в статье в качестве примера. У Миллера есть программа, которая отыскивает всевозможные лестницы между словами. Она написана для персонального компьютера фирмы IBM.

Очень интересная разновидность лестницы слов была предложена П. Блассом из Флорхем-Парка (шт. Нью-Джерси). Вдобавок к замене одной буквы еще разрешается словам ползать, подобно червякам, уменьшаясь или удлиняясь на одну букву в начале или в конце слова.

Воспользовавшись одним или двумя промежуточными словами (такими, как *orad*), Дж. Эрхард из Ричмонда (шт. Виргиния) нашел способ прев-

WWW.SCIAM-MAGAZINE.NAROD.RU



ратить зло (*evil*) в добро (*good*). Д. Фарр из Фрэмингхема (шт. Массачусетс) продемонстрировал лестницу слов от *solid* (сплошной) к *plane* (плоский). По пути он проделал кувырки (*flips*) и скольжение (*slide*).

В задаче о марсианском тексте мы имеем дело с текстовыми операциями другого рода. При заданном словаре, в котором указаны все дозволенные подстановки одних цепочек вместо других, нужно преобразовать исходное слово в конечное. Задача была решена А. Перту III из Сиэтла (шт. Вашингтон) и Э. Гротом из Скотсдэйла (шт. Аризона). Она легко решается после того, как вы догадаетесь, что несколько элементов словаря означают простые правила, при помощи которых можно манипулировать символами текста.

Очень забавны вычислительные фокусы со словами, придуманные Р. Хардином. О них шла речь в октябрьском номере журнала. Читатели, наверное, помнят странное стихотворение, начинающееся словами «Tweeze Denied Beef Worker Istmuth» (бессмысленное сочетание слов. — *Ред.*), которое по звучанию удивительно напоминает знакомое всем рождественское стихотворение К.К. Мура. Искусством трансформации поэтических произведений уже давно овладели люди, не обладавшие компьютерами. Р. Рид, математик, увлекающийся играми, из Университета Ватерлоо в Онтарио рассказал мне о забавной французской книжонке, содержащей всевозможные искажения известных стихотворений.

В английской литературе также существуют образчики подобного рода. Двое наших читателей прислали очаровательное стихотворение под названием «Ladle Rat Rotten Hat» (Очевидно, это искаженное название сказки о Красной Шапочке: Little Red Riding Hood — дословно: «маленький красный капюшончик». В искаженной форме что-то вроде: «Хлебай из гнилой крысиной норы». — *Ред.*). Действительно, для таких произведений компьютер едва ли необходим. Д. Вуд из Монреаля (пров. Квебек, Канада) написал импровизацию на тему песенки «It Came upon the Midnight Clear», звучащую следующим образом: «Eat cane a pond am I'd knight a glare» (бессмысленное сочетание слов, которое при беглом чтении очень созвучно исходной строке. — *Ред.*). Наконец, Р. Тилден из Соммервилла (шт. Массачусетс) напомнил мне, что известный американский карикатурист Уолт Келли дал нескольким своим произведениям шутливые названия такого же рода.

## *Где начинается нить судьбы?*

У позвоночных животных зародыш через несколько часов после оплодотворения представляет собой ровный шарик из трех слоев клеток — эктодермы, мезодермы и эндодермы. Наружный слой — эктодерма — образован внешне идентичными клетками. Если эти клетки растить одни в культуре, они превращаются в клетки кожи; собственно, и в организме большей части эктодермальных клеток предназначено стать кожей. Но на одном из самых ранних этапов эмбрионального развития в процессе так называемой гаструляции полоска эктодермальных клеток испытывает воздействие со стороны среднего слоя — мезодермы, и эти клетки, дифференцируясь, образуют особую структуру — нервную пластинку. Нервная пластинка, которая вначале выглядит лишь темной полосой на будущей спинной стороне все еще шарообразного зародыша, является предшественником головного и спинного мозга. Чем обусловлено ее появление?

Долгое время считалось, что индукция (т. е. внешнее влияние) со стороны мезодермы побуждает определенные эктодермальные клетки дифференцироваться с образованием нервной ткани. Для этого требуется непосредственный контакт между эктодермой и мезодермой. Но недавно получены данные, свидетельствующие, что внутренний фактор — программа, заложенная в самих эктодермальных клетках, — также играет определенную роль. Как сообщают К. Шарп и Дж. Гердон из Кембриджского университета, а также А. Фриц и Э. Де Робертис из Медицинской школы Калифорнийского университета в Лос-Анджелесе, изучавшие эмбриональное развитие африканской шпорцевой лягушки *Xenopus*, некоторые эктодермальные клетки еще до того, как они вступают в контакт с мезодермой, обладают особым «предрасположением» к превращению в нервную ткань.

Решающим шагом в этой работе было выделение фрагмента ДНК, экспрессия которого свойственна клеткам развивающейся нервной пластинки. Де Робертис отмечает, что этот фрагмент, вероятно, является не только специфическим маркером дифференцировки, но и ее инициатором. Чтобы подчеркнуть роль обнаруженного участка ДНК, исследователи назвали его гомеобоксом. Этот термин впервые был применен в отношении последовательности ДНК, определяющей ранние этапы эмбрионального развития плодовой мушки *Drosophila*.

Исследователи брали эктодермальные клетки из различных участков зародыша *Xenopus* до начала дифференцировки и инкубировали их с мезодермальными клетками. Оказалось, что те эктодермальные клетки, которые находились в зародыше на будущей спинной стороне, обладали большей чувствительностью к индукции, чем клетки с брюшной стороны: экспрессия гомеобокса в «спинных» клетках наблюдалась чаще. По мнению Гердона, это означает, что еще до гаструляции в эктодермальных клетках спинной стороны происходят изменения, подготавливающие их к превращению в нервную ткань; возможно, здесь дело в какой-то еще более ранней индукции. Как полагает Де Робертис, у млекопитающих, в том числе у человека, дифференцировка эктодермы тоже управляется и индукцией, и внутренними факторами. Он подчеркивает, что геном человека содержит последовательности, похожие на гомеобоксы *Xenopus* и *Drosophila*. Подтвердить это предположение экспериментально, однако, будет трудно, так как у млекопитающих зародыш на первых стадиях развития в тысячи раз меньше, чем у *Xenopus*.

ОДНОДНЕВНЫЙ ЗАРОДЫШ африканской шпорцевой лягушки уже имеет зачаток нервной системы — нервную пластинку в виде темной полосы. (Микрофотография Дж. Гердона.)